15437-0601/SUN040017NP *Patent*

UNITED STATES PATENT APPLICATION

FOR

MULTI-LEVEL RESOURCE LIMITS FOR OPERATING SYSTEM PARTITIONS

INVENTORS:

OZGUR C. LEONARD
ANDREW G. TUCKER
STEPHEN C. HAHN

PREPARED BY:

HICKMAN PALERMO TRUONG & BECKER LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125
(408) 414-1080

"Express Mail" mailing label number EV323351618US

Date of Deposit February 3, 2004

MULTI-LEVEL RESOURCE LIMITS FOR OPERATING SYSTEM PARTITIONS

RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/469,558, filed May 9, 2003, entitled OPERATING SYSTEM VIRTUALIZATION by Andrew G. Tucker, et al., the entire contents of which are incorporated herein by this reference.

BACKGROUND

[0002] In many computer implementations, it is desirable to be able to specify what amount of resources may be consumed by which entities. For example, it may be desirable to specify that a certain group of applications is allowed to consume an X amount of a set of resources, while another group of applications is allowed to consume a Y amount of the resources. This ability to allocate resources to specific entities enables a system administrator to better control how the resources of a system are used. This control may be used in many contexts to achieve a number of desirable results, for example, to prevent certain processes from consuming an inordinate amount of resources, to enforce fairness in resource usage among various entities, to prioritize resource usage among different entities, etc. Current systems allow certain resources to be allocated to certain entities. For example, it is possible to limit the total consumption of shared memory by certain groups of processes. However, the level of control that is possible with current systems is fairly limited.

SUMMARY

[0003] In accordance with one embodiment of the present invention, there is provided a mechanism for implementing multi-level resource control in operating system partitions. With this mechanism, it is possible to control resource allocation at multiple levels of an operating system environment.

[0004] In one embodiment, one or more non-global partitions may be established within a global operating system environment provided by an operating system. Each non-global partition serves to isolate the processes running within that partition from the other non-global partitions within the global operating system environment. In one embodiment, each non-global partition may have one or more projects executing therein, and each project may comprise one or more processes.

[0005] Each non-global partition may have associated therewith a first resource limit. This resource limit indicates a maximum amount of a particular resource (e.g. shared memory) that can be allocated to the non-global partition as a whole. In one embodiment, the first resource limit is assigned by a global administrator. By specifying a resource limit for a non-global partition, the global administrator is in effect specifying how much of that resource is available to all of the projects and processes within that non-global partition.

[0006] In one embodiment, each project executing within a non-global partition may have associated therewith a second resource limit. This limit indicates what amount of the particular resource that can be allocated to the non-global partition can be allocated to that project. In one embodiment, the second resource limit is assigned by a non-global administrator responsible for administering the non-global partition.

[0007] From the above discussion, it is clear that this embodiment of the present invention enables the allocation of the particular resource to be controlled at multiple levels.

More specifically, the global administrator can control what amount of the resource can be allocated to a non-global partition as a whole, and the non-global administrator can control how that amount of the particular resource can be allocated to the various projects within that non-global partition. This ability to control resource allocation at multiple levels makes it possible to exercise better control over how resources are used in a computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] FIG. 1 is a functional diagram of an operating system environment comprising a global zone and one or more non-global zones, in accordance with one embodiment of the present invention;

[0009] FIG. 2 is a functional diagram of an operating system environment comprising a global zone and one or more non-global zones containing projects and sharing resources, in accordance with one embodiment of the present invention;

[0010] FIG. 3 is a functional diagram that graphically illustrates the composition of a project, in accordance with one embodiment of the present invention;

[0011] FIG. 4 is a functional diagram that graphically illustrates the relationship between zone resource limits and project resource limits, where a zone limit overrides a project limit, in accordance with one embodiment of the present invention;

[0012] FIG. 5 is a functional diagram that illustrates a task level viewpoint of one embodiment of the present invention that services process resource allocation requests, in accordance with one embodiment of the present invention;

[0013] FIG. 6 is a functional diagram that graphically illustrates a project within a zone requesting a shared memory allocation with project and zone shared memory limits imposed, in accordance with one embodiment of the present invention;

[0014] FIG. 7 is a functional diagram that graphically illustrates two projects within a zone requesting a shared memory allocation with project and zone shared memory limits imposed, in accordance with one embodiment of the present invention;

[0015] FIG. 8 is a block diagram that illustrates a computer system upon which an embodiment may be implemented;

[0016] FIG. 9 is an operational flow diagram, which provides a high level overview of one embodiment of the present invention; and

[0017] FIG. 10 is a flowchart illustrating the determination of computing resource allocation to a process, in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF EMBODIMENTS(S)

Conceptual Overview

[0018] In accordance with one embodiment of the present invention, there is provided a mechanism for implementing multi-level resource control in operating system partitions. With this mechanism, it is possible to control resource allocation at multiple levels of an operating system environment. An operational flow diagram, which provides a high level overview of this embodiment of the present invention, is shown in Fig. 9.

[0019] In one embodiment, one or more non-global partitions may be established (block 902) within a global operating system environment provided by an operating system. Each non-global partition serves to isolate the processes running within that partition from the other non-global partitions within the global operating system environment. In one embodiment, each non-global partition may have one or more projects executing therein, and each project may comprise one or more processes.

[0020] Each non-global partition may have associated (block 904) therewith a first resource limit. This resource limit indicates a maximum amount of a particular resource (e.g. shared memory) that can be allocated to the non-global partition as a whole. In one embodiment, the first resource limit is assigned by a global administrator. By specifying a resource limit for a non-global partition, the global administrator is in effect specifying how much of that resource is available to all of the projects and processes within that non-global partition.

[0021] In one embodiment, each project executing within a non-global partition may have associated (block 906) therewith a second resource limit. This limit indicates what amount of the particular resource that can be allocated to the non-global partition can be

allocated to that project. In one embodiment, the second resource limit is assigned by a non-global administrator responsible for administering the non-global partition.

[0022] From the above discussion, it is clear that this embodiment of the present invention enables the allocation of the particular resource to be controlled at multiple levels. More specifically, the global administrator can control what amount of the resource can be allocated to a non-global partition as a whole, and the non-global administrator can control how that amount of the particular resource can be allocated to the various projects within that non-global partition. This ability to control resource allocation at multiple levels makes it possible to exercise better control over how resources are used in a computer system.

System Overview

[0023] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

[0024] Fig. 1 illustrates a functional block diagram of an operating system (OS) environment 100 in accordance with one embodiment of the present invention. OS environment 100 may be derived by executing an OS in a general-purpose computer system, such as computer system 800 illustrated in Fig. 8, for example. For illustrative purposes, it will be assumed that the OS is Solaris manufactured by Sun Microsystems, Inc. of Santa Clara, California. However, it should be noted that the concepts taught herein may be applied to any OS, including but not limited to Unix, Linux, Windows, MacOS, etc.

[0025] As shown in Fig. 1, OS environment 100 may comprise one or more zones (also referred to herein as partitions), including a global zone 130 and zero or more non-global zones 140. The global zone 130 is the general OS environment that is created when the OS is booted and executed, and serves as the default zone in which processes may be executed if no non-global zones 140 are created. In the global zone 130, administrators and/or processes having the proper rights and privileges can perform generally any task and access any device/resource that is available on the computer system on which the OS is run. Thus, in the global zone 130, an administrator can administer the entire computer system. In one embodiment, it is in the global zone 130 that an administrator executes processes to configure and to manage the non-global zones 140.

[0026] The non-global zones 140 represent separate and distinct partitions of the OS environment 100. One of the purposes of the non-global zones 140 is to provide isolation. In one embodiment, a non-global zone 140 can be used to isolate a number of entities, including but not limited to processes 170, one or more file systems 180, and one or more logical network interfaces 182. Because of this isolation, processes 170 executing in one non-global zone 140 cannot access or affect processes in any other zone. Similarly, processes 170 in a non-global zone 140 cannot access or affect the file system 180 of another zone, nor can they access or affect the network interface 182 of another zone. As a result, the processes 170 in a non-global zone 140 are limited to accessing and affecting the processes and entities in that zone. Isolated in this manner, each non-global zone 140 behaves like a virtual standalone computer. While processes 170 in different non-global zones 140 cannot access or affect each other, it should be noted that they may be able to communicate with each other via a network connection through their respective logical network interfaces 182.

This is similar to how processes on separate standalone computers communicate with each other.

[0027] Having non-global zones 140 that are isolated from each other may be desirable in many applications. For example, if a single computer system running a single instance of an OS is to be used to host applications for different competitors (e.g. competing websites), it would be desirable to isolate the data and processes of one competitor from the data and processes of another competitor. That way, it can be ensured that information will not be leaked between the competitors. Partitioning an OS environment 100 into non-global zones 140 and hosting the applications of the competitors in separate non-global zones 140 is one possible way of achieving this isolation.

[0028] In one embodiment, each non-global zone 140 may be administered separately. More specifically, it is possible to assign a zone administrator to a particular non-global zone 140 and grant that zone administrator rights and privileges to manage various aspects of that non-global zone 140. With such rights and privileges, the zone administrator can perform any number of administrative tasks that affect the processes and other entities within that non-global zone 140. However, the zone administrator cannot change or affect anything in any other non-global zone 140 or the global zone 130. Thus, in the above example, each competitor can administer his/her zone, and hence, his/her own set of applications, but cannot change or affect the applications of a competitor. In one embodiment, to prevent a non-global zone 140 from affecting other zones, the entities in a non-global zone 140 are generally not allowed to access or control any of the physical devices of the computer system.

[0029] In contrast to a non-global zone administrator, a global zone administrator with proper rights and privileges may administer all aspects of the OS environment 100 and the

computer system as a whole. Thus, a global zone administrator may, for example, access and control physical devices, allocate and control system resources, establish operational parameters, etc. A global zone administrator may also access and control processes and entities within a non-global zone 140.

[0030] In one embodiment, enforcement of the zone boundaries is carried out by the kernel 150. More specifically, it is the kernel 150 that ensures that processes 170 in one non-global zone 140 are not able to access or affect processes 170, file systems180, and network interfaces 182 of another zone (non-global or global). In addition to enforcing the zone boundaries, kernel 150 also provides a number of other services. These services include but are certainly not limited to mapping the network interfaces 182 of the non-global zones 140 to the physical network devices 120 of the computer system, and mapping the file systems 180 of the non-global zones 140 to an overall file system and a physical storage 110 of the computer system. The operation of the kernel 150 will be discussed in greater detail in a later section.

Non-Global Zone States

[0031] In one embodiment, a non-global zone 140 may take on one of four states: (1) Configured; (2) Installed; (3) Ready; and (4) Running. When a non-global zone 140 is in the Configured state, it means that an administrator in the global zone 130 has invoked an operating system utility (in one embodiment, zonecfg(1m)) to specify all of the configuration parameters of a non-global zone 140, and has saved that configuration in persistent physical storage 110. In configuring a non-global zone 140, an administrator may specify a number of different parameters. These parameters may include, but are not limited to, a zone name, a zone path to the root directory of the zone's file system 180, specification of one or more file

systems to be mounted when the zone is created, specification of zero or more network interfaces, specification of devices to be configured when the zone is created, resource limits for the zone, and zero or more resource pool associations.

[0032] Once a zone is in the Configured state, a global administrator may invoke another operating system utility (in one embodiment, zoneadm(1m)) to put the zone into the Installed state. When invoked, the operating system utility interacts with the kernel 150 to install all of the necessary files and directories into the zone's root directory, or a subdirectory thereof.

[0033] To put an Installed zone into the Ready state, a global administrator invokes an operating system utility (in one embodiment, zoneadm(1m) again), with a zoneadmd process 162 to be started (there is a zoneadmd process associated with each non-global zone). In one embodiment, zoneadmd 162 runs within the global zone 130 and is responsible for managing its associated non-global zone 140. After zoneadmd 162 is started, it interacts with the kernel 150 to establish the non-global zone 140. In establishing a non-global zone 140, a number of operations may be performed, including but not limited to assigning a zone ID, starting a zsched process 164 (zsched is a kernel process; however, it runs within the non-global zone 140, and is used to track kernel resources associated with the non-global zone 140), mounting file systems 180, plumbing network interfaces 182, configuring devices, and setting resource controls. These and other operations put the non-global zone 140 into the Ready state to prepare it for normal operation.

[0034] Putting a non-global zone 140 into the Ready state gives rise to a virtual platform on which one or more processes may be executed. This virtual platform provides the infrastructure necessary for enabling one or more processes to be executed within the non-global zone 140 in isolation from processes in other non-global zones 140. The virtual platform also makes it possible to isolate other entities such as file system 180 and network

interfaces 182 within the non-global zone 140, so that the zone behaves like a virtual standalone computer. Notice that when a non-global zone 140 is in the Ready state, no user or non-kernel processes are executing inside the zone (recall that zsched is a kernel process, not a user process). Thus, the virtual platform provided by the non-global zone 140 is independent of any processes executing within the zone. Put another way, the zone and hence, the virtual platform, exists even if no user or non-kernel processes are executing within the zone. This means that a non-global zone 140 can remain in existence from the time it is created until either the zone or the OS is terminated. The life of a non-global zone 140 need not be limited to the duration of any user or non-kernel process executing within the zone.

[0035] After a non-global zone 140 is in the Ready state, it can be transitioned into the Running state by executing one or more user processes in the zone. In one embodiment, this is done by having zoneadmd 162 start an init process 172 in its associated zone. Once started, the init process 172 looks in the file system 180 of the non-global zone 140 to determine what applications to run. The init process 172 then executes those applications to give rise to one or more other processes 174. In this manner, an application environment is initiated on the virtual platform of the non-global zone 140. In this application environment, all processes 170 are confined to the non-global zone 140; thus, they cannot access or affect processes, file systems, or network interfaces in other zones. The application environment exists so long as one or more user processes are executing within the non-global zone 140.

[0036] After a non-global zone 140 is in the Running state, its associated zoneadmd 162 can be used to manage it. Zoneadmd 162 can be used to initiate and control a number of zone administrative tasks. These tasks may include, for example, halting and rebooting the non-global zone 140. When a non-global zone 140 is halted, it is brought from the Running

state down to the Installed state. In effect, both the application environment and the virtual platform are terminated. When a non-global zone 140 is rebooted, it is brought from the Running state down to the Installed state, and then transitioned from the Installed state through the Ready state to the Running state. In effect, both the application environment and the virtual platform are terminated and restarted. These and many other tasks may be initiated and controlled by zoneadmd 162 to manage a non-global zone 140 on an ongoing basis during regular operation.

Multi-Level Resource Control

[0037] As noted previously, one embodiment of the present invention allows resource allocation limits to be set at multiple levels. In one embodiment, a global zone administrator (also referred to herein as a global administrator) can set zone resource limits at a zone level. A non-global zone administrator (also referred to herein as a non-global administrator) can set project resource limits at a project level within a zone. As used herein, the term project refers broadly to a group of one or more processes running within a zone.

[0038] Fig. 2 illustrates a functional diagram of the OS environment 100 with zones 140 sharing resources 201. As noted above, zones contain processes that are executed by an init process. The global and non-global zone administrators have the ability to define an abstract called a project in a zone to track processes. Each zone 130, 140 can contain one or more projects. In this example, zone A 140(a) contains Project 1 202 and Project 2 203, zone B 140(b) contains Project 3 204 and Project 4 205, and the global zone 130 contains Project 5 206.

[0039] In one embodiment, a global zone administrator is allowed to set resource limits for zones 140. Limits can be set for any resource that is available to a zone, *e.g.*, locked

memory, number of active processes, processor share, number of semaphores, ports, message queues, etc. A non-global zone administrator is allowed to set resource limits for projects 202-205 within a zone 140. The non-global zone administrator can look at the zone limits that have been set by the global zone administrator, but cannot modify the values. A global zone administrator can set resource limits for projects 206 in the global zone.

[0040] Fig. 3 illustrates the composition of a project in one embodiment. The project 301 is composed of one or more tasks 302. Each task 302 is further comprised of processes 302. A project is a member of only one zone. Projects and tasks are levels of abstraction that allow a non-global zone administrator to logically group one or more processes. As each process runs, it makes requests for resources to the kernel 150. The kernel 150 keeps track of which process is part of which project. A non-global zone administrator can further refine his resource limits down to the task and process level, if desired. For readability reasons, resource limits at the project level will be discussed herein with the understanding that task and process resource limits are a natural extension of the method described.

[0041] When a process within a project makes a resource request, the kernel 150 checks the project's resource limit to see if the resource can be allocated to the process. If the project's resource usage is less than the project's limit, then the kernel 150 may be able to allocate the resource to the process up to the project limit. However, before any allocation occurs, the kernel 150 first checks the zone limit of the zone in which the project is executing. Even though granting the request would keep the consumption below the project limit, it may cause it to go over its zone-level resource limit. If the zone is below its resource limit and the project is below its resource limit, then the kernel can allocate the resource to the process up to the project limit or zone limit, whichever is less.

[0042] Fig. 4 illustrates the interaction between a zone limit and a project limit. In this example, a zone 401 has a limit for a resource set to 100 by a global zone administrator. A project 402 within the zone has had its limit for the same resource set to 10 by a non-global zone administrator. If the project is below its limit when a process within the project makes a request for the resource, and the total allocation for the resource for all projects within the zone is below the zone limit, then the kernel can allocate the resource to the process up to whatever is left of the project's allocation or the zone's allocation, whichever is less.

[0043] Another example illustrates the control of zone limits. The zone 403 has a limit of 100 for a particular resource. A non-global zone administrator has set the limit for the resource at 1000 for a project 404 within the zone 403. When a process within the project makes a request for the resource, the kernel can only allocate the resource up to the lowest limit. Since the zone 403 has a limit of 100, the total resource allocation for all projects in the zone can only be 100. It does not matter what the non-global zone administrator has set the resource limits at for the projects within the zone because the total usage must be less than or equal to the zone limit. The zone limit overrides the limit that is set by the non-global zone administrator. This gives the global zone administrator control over errant or malicious non-global zone administrators.

[0044] Fig. 5 is a task viewpoint of one embodiment of the present invention. The kernel 501 (which coincides with kernel 150 of Fig. 1) records zone resource limits set by the global zone administrator, and project resource limits set by the non-global zone administrators in the zone settings storage 503. The kernel 501 services requests for resources made by running processes.

[0045] When a process makes a resource request, the kernel 501 passes the request to the check limits module 502. The check limits module 502 looks up the resource limits for the

process' project and zone in the zone settings storage 503. The check limits module 502 will approve the process' resource request only if the project is below its limit and the zone is also below its limit for that particular resource. As shown in Fig. 10, the amount of the resource that is approved depends on the project and zone limits. The amount of the resource that is available for allocation is determined by the amount that remains of the project allocation and the amount that remains of the zone allocation. The lower of the two values indicates the available amount that can be allocated to the process 1001. If the process' request is less than or equal to the available amount 1002, then the process receives the full amount of the resource that it is requesting 1004. If the process' request is greater than the available amount 1002, then the process will receive nothing, resulting in a denied request 1003. Optionally, if the process' request is greater than the available amount, then the process will only receive the available amount.

[0046] In one embodiment, the check limits module 502 records a running total of the amount of the resource that is allocated to each project and each zone in the zone settings storage 503. The totals are updated upon allocation of a resource. This allows the check limits module 502 to respond faster to the kernel 501.

[0047] Fig. 6 illustrates an example where a zone shared memory limit for zone A 601 has been set to 20000 bytes, and the project shared memory limit for a project 1 602 within zone A 601 has been set to 10000 bytes. A process within project P1 602 makes a request for one byte of shared memory. The kernel checks the amount of the resource that has already been allocated to project P1 602 and finds that 10000 bytes have already been allocated to project P1 602. The kernel checks the project shared memory limit and sees that the project shared memory limit is 10000 bytes. The kernel compares the project shared memory limit

with the amount that has already been allocated to the project and finds that the project is already at its limit. The kernel denies the process' request.

[0048] Fig. 7 illustrates an example where a zone shared memory limit for zone B 701 has been set to 20000 bytes, and the project shared memory limits for projects P2 702 and P3 703 within zone B 701 have been set to 10000 bytes each. A process within project P2 702 makes a request for one byte of shared memory. The kernel checks the amount of the resource that has already been allocated to project P2 702 and finds that 5000 bytes have already been allocated to project P2 702. The kernel checks the project shared memory limit and sees that the project shared memory limit is 10000 bytes. The kernel compares the project shared memory limit with the amount that has already been allocated to project P2 702 and finds that 5000 bytes can still be allocated to the project.

[0049] The kernel then calculates the total amount of shared memory that has been allocated to all active projects in zone B 701. The kernel adds the amount allocated to project P2 702 and project P3 703 and finds that 10000 bytes have already been allocated to zone B 701 as a whole. Comparing that value with the zone shared memory limit, the kernel finds that 10000 bytes can still be allocated to zone B 701.

[0050] The lower of the available project and zone amounts is the project amount, which is 5000 bytes. The kernel approves the process' request because the one byte request is less than 5000 bytes. The kernel can record the total shared memory allocated to projects and zones each time an allocation is made to save the time consumed by performing dynamic calculations at each allocation.

[0051] Suppose now that, rather than 5000 bytes, 15000 bytes have been allocated to project P3 703 (suppose further that the project limit for project P3 is 15000 bytes, not 10000 bytes). This would mean that the total shared memory already allocated to zone B as a whole

has reached the zone limit of 20000 bytes. In such a case, even though the project limit has not been reached, the process' request will be denied because the zone limit has been reached.

HARDWARE OVERVIEW

[0052] Fig. 8 is a block diagram that illustrates a computer system 800 upon which an embodiment of the invention may be implemented. Computer system 800 includes a bus 802 for facilitating information exchange, and one or more processors 804 coupled with bus 802 for processing information. Computer system 800 also includes a main memory 806, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 802 for storing information and instructions to be executed by processor 804. Main memory 806 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 804. Computer system 800 may further include a read only memory (ROM) 808 or other static storage device coupled to bus 802 for storing static information and instructions for processor 804. A storage device 810, such as a magnetic disk or optical disk, is provided and coupled to bus 802 for storing information and instructions.

[0053] Computer system 800 may be coupled via bus 802 to a display 812, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 814, including alphanumeric and other keys, is coupled to bus 802 for communicating information and command selections to processor 804. Another type of user input device is cursor control 816, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 804 and for controlling cursor movement on display 812. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0054] In computer system 800, bus 802 may be any mechanism and/or medium that enables information, signals, data, etc., to be exchanged between the various components. For example, bus 802 may be a set of conductors that carries electrical signals. Bus 802 may also be a wireless medium (e.g. air) that carries wireless signals between one or more of the components. Bus 802 may also be a medium (e.g. air) that enables signals to be capacitively exchanged between one or more of the components. Bus 802 may further be a network connection that connects one or more of the components. Overall, any mechanism and/or medium that enables information, signals, data, etc., to be exchanged between the various components may be used as bus 802.

[0055] Bus 802 may also be a combination of these mechanisms/media. For example, processor 804 may communicate with storage device 810 wirelessly. In such a case, the bus 802, from the standpoint of processor 804 and storage device 810, would be a wireless medium, such as air. Further, processor 804 may communicate with ROM 808 capacitively. In this instance, the bus 802 would be the medium (such as air) that enables this capacitive communication to take place. Further, processor 804 may communicate with main memory 806 via a network connection. In this case, the bus 802 would be the network connection. Further, processor 804 may communicate with display 812 via a set of conductors. In this instance, the bus 802 would be the set of conductors. Thus, depending upon how the various components communicate with each other, bus 802 may take on different forms. Bus 802, as shown in Fig. 8, functionally represents all of the mechanisms and/or media that enable information, signals, data, etc., to be exchanged between the various components.

[0056] The invention is related to the use of computer system 800 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 800 in response to processor 804 executing

one or more sequences of one or more instructions contained in main memory 806. Such instructions may be read into main memory 806 from another machine-readable medium, such as storage device 810. Execution of the sequences of instructions contained in main memory 806 causes processor 804 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0057] The term "machine-readable medium" as used herein refers to any medium that participates in providing data that causes a machine to operation in a specific fashion. In an embodiment implemented using computer system 800, various machine-readable media are involved, for example, in providing instructions to processor 804 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 810. Volatile media includes dynamic memory, such as main memory 806. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 802. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0058] Common forms of machine-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0059] Various forms of machine-readable media may be involved in carrying one or more sequences of one or more instructions to processor 804 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 800 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 802. Bus 802 carries the data to main memory 806, from which processor 804 retrieves and executes the instructions. The instructions received by main memory 806 may optionally be stored on storage device 810 either before or after execution by processor 804.

[0060] Computer system 800 also includes a communication interface 818 coupled to bus 802. Communication interface 818 provides a two-way data communication coupling to a network link 820 that is connected to a local network 822. For example, communication interface 818 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 818 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 818 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0061] Network link 820 typically provides data communication through one or more networks to other data devices. For example, network link 820 may provide a connection through local network 822 to a host computer 824 or to data equipment operated by an

Internet Service Provider (ISP) 826. ISP 826 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 828. Local network 822 and Internet 828 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 820 and through communication interface 818, which carry the digital data to and from computer system 800, are exemplary forms of carrier waves transporting the information.

[0062] Computer system 800 can send messages and receive data, including program code, through the network(s), network link 820 and communication interface 818. In the Internet example, a server 830 might transmit a requested code for an application program through Internet 828, ISP 826, local network 822 and communication interface 818.

[0063] The received code may be executed by processor 804 as it is received, and/or stored in storage device 810, or other non-volatile storage for later execution. In this manner, computer system 800 may obtain application code in the form of a carrier wave.

[0064] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.
